第3章　空間分割多重の信号処理をFPGAに実装する

スマート・アンテナのビーム・フォーミング技術

Minseok Kim

スマート・アンテナ技術はIEEE 802.11nで注目が高まるMIMO(Multiple Input Multiple Output)技術とともに，3GPPや3GPP2，IEEE 802.16，IEEE 802.11など，さまざまな通信システムで次々と取り入れられると考えられる．本章では，スマート・アンテナ・システムの基本原理と信号処理の実装について説明する．（筆者）

近年，音声通話以外に高速なデータ・サービスへの期待が高まっています．プロバイダ側では，基地局で収容可能なユーザ数を増やしながら設備費用を節減し，適切なサービスを開発しなければなりません．

無線システムでデータ伝送の高速化を実現するには，制限された周波数資源の利用効率を向上する必要があります．このために導入されたのがスマート・アンテナ・システム(Smart Antenna System)注1です．

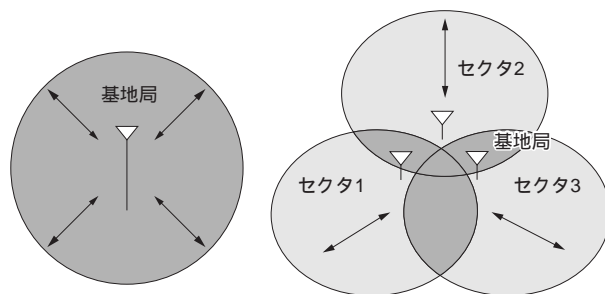
注1：アダプティブ・アレイ・アンテナ(Adaptive Array Antenna)とも呼ぶ．

(a) 無指向性アンテナ　(b) 120°，3セクタ・アンテナ

図1　従来の基地局アンテナ

1．スマート・アンテナとは

● スマート・アンテナの動作イメージ

割り当てられる周波数帯の不足により，制限された周波数帯域内で通信容量を増やすことが大きな課題となっています．

図1(a)のように，全方向に対して指向性を持たないアンテナ基地局では，各ユーザ信号の送受信は，同一周波数を利用するセル内のほかのユーザにそのまま干渉してしまいます．不要な方向に無駄な電力を放射するため，そのシステム性能が干渉電力により大きく制限されることになります．この問題を解決するために工夫したのがセクタ・アンテナです．セクタ・アンテナとは，指向性を持つアンテナを用いてカバー領域を分割します．同一セル内において，異なるセクタ間の干渉を抑えることができ，複数のユーザを収容できる基地局アンテナ・システムです．**図1(b)**は3セクタ・アンテナの例を示しています．

スマート・アンテナは，空間信号処理により同一周波数の干渉を抑えることができ，制限された周波数でシステム容量の増大をはかる技術です．あるユーザ方向にアンテナ指向性のメイン・ビームを向け，ほかのユーザにはヌルを形成し送受信を行うことで，干渉の影響を取り除けます．この方式は空間的にチャネルを分割するので，空間分割多重方式(SDMA：Space Division Multiple Access)と呼ばれます．**図2**にその概要を示します．

周波数の利用効率を上げるために，距離の離れた場所(異なるセル)では周波数を繰り返し利用しています．スマー

KeyWord　スマート・アンテナ，アダプティブ・アレイ・アンテナ，空間分割多重方式，適応ビーム形成，シストリック・アレイ

図2 スマート・アンテナを用いた空間分割多重(SDMA：Space Division Multiple Access)の概念図

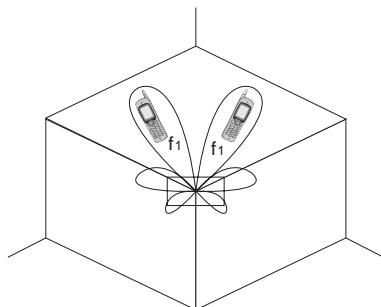
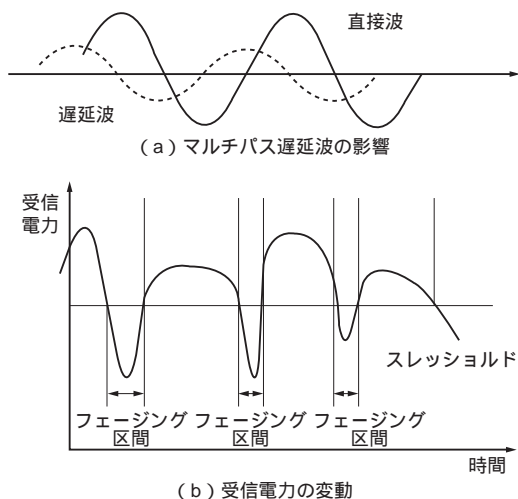
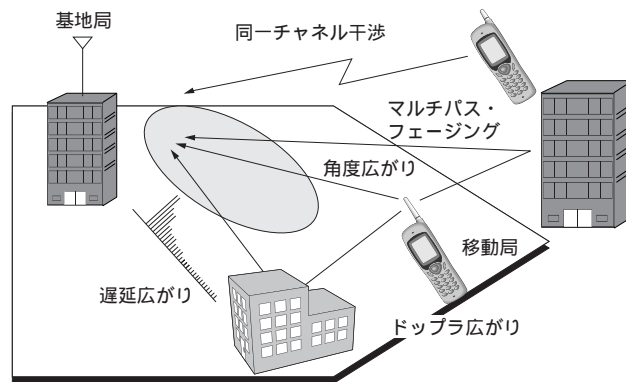
図3 移動通信における伝搬状況

ト・アンテナ技術を用いると，ある条件(方向)に対して送信電力を制御できるため，隣接セルにおける干渉電力を大幅に抑えられます．

図3に，携帯電話のセルラ基地局における伝搬環境のイメージを示します．伝搬路の状況によりさまざまな現象が起きます．建物などにより電波が反射，回折，散乱が起き，複数の伝搬路を通過するために生じるマルチパス・フェージング(Multipath Fading)や，同一周波数を利用している隣接のセルのユーザによる同一チャネル干渉(CCI：Co-channel Interference)，伝搬遅延時間差(遅延広がり特性)に起因するシンボル間干渉(ISI：Inter-symbol Interference)によって通信容量や品質が劣化してしまいます．

例えば，受信アンテナに到来する信号は，**図4(a)**のようにマルチパスにより，それぞれある位相差を持って辿り着きます．その信号の和は位相差によって強め合ったり，弱め合ったりします．このため，**図4(b)**のように受信電力が激しく変化し，時間あるいは場所により通信状況が劣化します．最悪は途切れてしまうことになります．これをマルチパス・フェージングと呼びます．

マルチパス・フェージングを含めた諸問題に対して，以前から送信電力制御や誤り制御符号，ダイバシティ技術が導入されていました．そして，周波数利用効率をより高めるために，スマート・アンテナを用いて干渉波や遅延波の除去をすることを目的とした研究が盛んに行われました．

スマート・アンテナの適応ビーム形成理論は，古典的な適応等価技術と等価です．最近は，送受信ともに複数のアンテナを考慮したMIMOシステムの研究にウェイトが移りつつあり，一層活発に研究開発が行われています．

スマート・アンテナ・システムは，**図5**のようにアレイ・アンテナと複数の無線チャネル，信号処理部で構成されています．ベースバンドにおけるディジタル信号処理により適応的なアンテナの指向性を制御できます．スマート・アンテナにおけるビーム形成は，伝搬路の状況に対して最適な複素重み係数を計算し，各アンテナ素子からの信号に重み係数を乗算します．これは，信号の振幅と位相の数値的

図4 マルチパス信号の影響

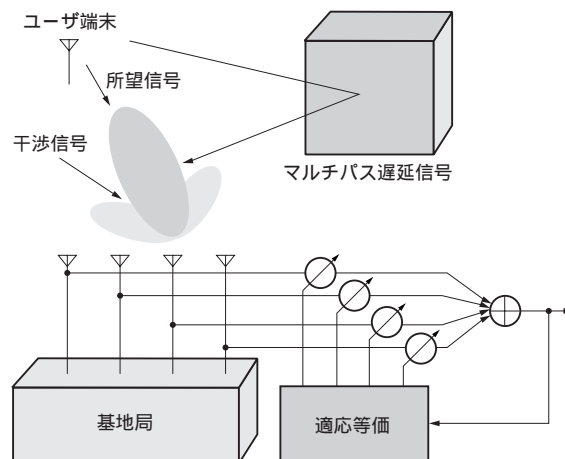



図5 セルラ・システムのスマート・アンテナの動作

な調整になります．この処理より，アレイ出力には，非常に品質の高い（最大の$SINR$：ノイズと干渉電力に対する所望信号電力の比）信号が得られます．

適応ビーム形成手法として，何通りかの固定ビーム・パターンをあらかじめ用意し，受信信号強度に基づいてビーム・パターンを切り替える方法を，スイッチド・ビーム（Switched Beam）形成といいます．一方，スマート・アンテナにおける適応ビーム形成（Adaptive Beamforming）は，伝搬路の状況に応じて最適な重みをリアルタイムで計算し，目的のユーザに向けてメイン・ビームを，干渉となるユーザに向けてヌルを形成できるので，信号の品質が改善されます．

● 適応ビーム形成によるヌル走査（ヌル・ステアリング）

スマート・アンテナは，アンテナ指向性パターンを適応的に制御できると説明しました．特にヌルの走査能力が大きな特徴となります．ヌルを走査することで干渉波の影響をうまく取り除けるからです．次に，指向性のパターンを制御する方法を簡単な例を通して紹介します[3]．

簡単のため，半波長間隔に配置された2素子アレイ・アンテナに二つの信号がそれぞれある角度から到来するモデルを考えます（**図6**）．仮に信号1は所望信号（s_1，入射角：θ_1）とし，信号2は干渉信号（s_2，入射角：θ_2）とします．そうすると，k番目のアンテナに受信される信号x_kは

$$x_k = \sum_{l=1}^{2} s_l(t) \cdot \exp\left(-j\frac{2\pi}{\lambda} d(k-1)\sin\theta_1\right) + n_k(t), \quad k = 1,2 \quad \cdots(1)$$

となります．ここで，$s(t)$は入射波の包絡線，λは波長，n_kは白色ガウス・ノイズです．さらに簡単のため，ノイズはないことにします．アンテナ間隔dを$\lambda/2$とすると，アレイ・アンテナの出力は，

$$\begin{aligned} y(t) &= \sum_{k=1}^{2} \omega_k^* x_k(t) \\ &= \sum_{l=1}^{2} \omega_1^* s_l(t) + \sum_{l=1}^{2} \omega_2^* s_l(t) \cdot \exp(-j\pi \sin\theta_l) \end{aligned} \quad \cdots\cdots(2)$$

となります．この場合，所望信号は受信し，干渉信号にヌルを向けるためには，以下の条件を満足すればよいことが分かります．

$$\begin{aligned} \omega_1 + \omega_2 \exp(j\pi \sin\theta_1) &= 1 \\ \omega_1 + \omega_2 \exp(j\pi \sin\theta_2) &= 0 \end{aligned} \quad \cdots\cdots(3)$$

例えば，θ_1が0 °，θ_2が－30 °のとき，解は，

$$\begin{aligned} \omega_1 &= 0.5 - 0.5j \\ \omega_2 &= 0.5 + 0.5j \end{aligned} \quad \cdots\cdots(4)$$

となります．

このとき，ビーム・パターンをプロットしたのが**図7**です．ここで－30 °方向にヌル（落ち込み）が形成されていることが分かります．このヌルの方向を制御することがスマート・アンテナにおける適応ビーム形成の基本原理です．

メイン・ビームが0 °方向からずれているのは，最大ビーム方向に対しては制御を行っていないためです．最大ビーム方向については，これから説明する平均二乗誤差最小化手法（MMSE：Minimizing Mean Square Error）により最大$SINR$が得られるように制御できます．

2．スマート・アンテナにおける最適化手法

ここで，スマート・アンテナにおける適応ビーム形成法とその最適化手法について説明します．スマート・アンテナの機能は，適応ビーム形成と適応ヌル・ステアリング（Adaptive Null-steering）があります．適応ビーム形成は，受信波の到来方向が未知，あるいは時間的に変化する場合にも，アレイのビームを自動的に追従させる機能です．

一方，強い妨害波の存在下で微弱な所望電波を受信する場合に一般的な指向性合成法を用いるとすれば，非常に低いサイドローブ・レベルを設定しなければなりません．そこで，指向性パターンのヌル点を自動的に妨害波方向に向ける必要性が生じます．これが，適応ヌル・ステアリング

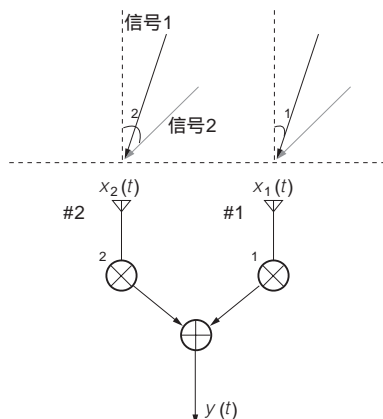



図6 2素子アレイと2波の信号モデル

という機能です．

スマート・アンテナは，電波環境に関する情報を学習しながら，指向特性および周波数特性を環境に合わせて変えていきます．不要波とノイズを含んだ電波環境から所望の信号を抽出するために，所望の信号に関する予備知識が必要です．通常，信号の中心周波数（搬送波周波数），到来波方向，変調方式，偏波などが利用されます．そのため，スマート・アンテナの動作原理は，それら予備知識および評価関数によって次のように大別されます(2)．

①平均二乗誤差最小化法（Minimum Mean Square Error: MMSE）
②最大SNR法（Maximum Signal-to-noise Ratio：MSN）
③拘束付出力電力最小化法（Constrained Minimization of Power：CMP）
④定包絡線信号用アルゴリズム（Constant Modulus Algorithm：CMA）

ここでは，最も広く用いられている平均二乗誤差最小化法（MMSE法）について簡単に説明します．

● 平均二乗誤差最小化法（MMSE）

MMSE法は，所望のアレイ応答である参照信号と実際のアレイ出力信号との誤差（誤差信号）を最小にすることによって，最適な重み係数を決定する方法です．この方法は適応ヌル・ステアリングと同時にアダプティブ・ビーム形成を行います．そのためにアンテナ素子配列やアレイ構成などに制約を受けないという長所がある一方，参照信号として厳密には所望の信号そのものを必要とするという矛盾があります．

実際には，所望の信号の性質（周波数帯域，変調方式など）に関する予備知識があるので，アレイ出力信号を適当に処理することによって適切な参照信号を得ることも可能です．従って，受信側で所望の信号のレプリカである参照信号を作るという概念は現実的な手段です．完全な所望信号の性質をあらかじめ既知の特性として解析を進められます．

最小化の対象となる誤差信号$e(t)$，すなわち，所望のアレイ応答（参照信号）$r(t)$と実際のアレイ出力信号$y(t)$との差は次式で与えられます．

$$e(t) = r(t) - y(t) = r(t) - \boldsymbol{\omega}^{\mathrm{H}} \cdot \boldsymbol{x}(t) \quad \cdots\cdots(5)$$

ここで，ωは信号合成に用いる重み係数です．上式より，誤差信号の2乗の期待値（平均二乗誤差）は次のように表されます．

$$E\left[\left|e(t)\right|^2\right] = E\left[\left|r(t) - y(t)\right|^2\right] = E\left[\left|r(t) - \boldsymbol{\omega}^{\mathrm{H}} \cdot \boldsymbol{x}(t)\right|^2\right] \cdots(6)$$

MMSE法の目的は重みベクトルωを適切に選ぶことによって上記の平均二乗誤差を最小にすることです．両辺に重みベクトルに対する勾配を取り，最小点を求める問題になります．この問題の解は，

$$\boldsymbol{\omega}_{opt} = \boldsymbol{R}_{xx}^{-1}\ \boldsymbol{r}_{xr} \quad \cdots\cdots(7)$$

となり，有名なウィーナ解（Wiener Solution）と呼ばれています．ここで，R_{xx}，r_{xr}はそれぞれ信号の自己相関行列，相関ベクトル（参照信号と入力信号ベクトルとの相関）になります．

$$\begin{aligned} \boldsymbol{R}_{xx} &\equiv \boldsymbol{E}\left[\boldsymbol{x}(t) \cdot \boldsymbol{x}^{\mathrm{H}}(t)\right] \\ \boldsymbol{r}_{xr} &\equiv \boldsymbol{E}\left[\boldsymbol{x}(t) \cdot \boldsymbol{r}^{*}(t)\right] \end{aligned} \quad \cdots\cdots(8)$$

● ディジタル制御アルゴリズム

MMSE法の最適な重み係数が以上のように与えられることがわかりました．これを観測可能なデータから求めるために，近似的な計算を用いた最適化アルゴリズムが必要となります．ディジタル制御による最適化アルゴリズムとしては，最急降下法に基づくLMS（Least Mean Square）アルゴリズムと，サンプル値を用いた直接解法のSMI（Sample Matrix Inversion）アルゴリズム，さらには再帰的最小二乗法のRLS（Recursive Least-squares）アルゴリズムが広く利用されています．

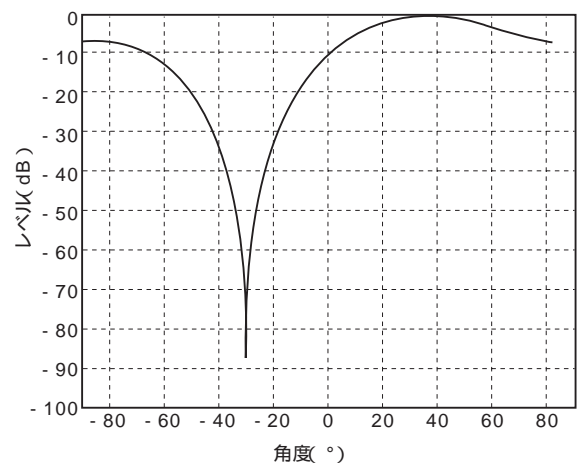



図7　適応ビーム形成による指向性パターンの一例
−30°方向にヌル（落ちこみ）が形成されているが，所望方向（0°方向）にビームがずれていることが確認できる

その中で，最急降下法は最も一般的です．確実に評価関数の最小点にたどりつくこと，計算負荷が小さいことなどが特徴です．しかし，最急降下法は入射波の到来角が接近していたり，各波の電力比が大きかったりする場合，収束が極端に遅くなるという欠点があることが知られています．この問題を克服する方法の一つがSMI方式です．

SMI方式は計算負荷が大きいことから従来敬遠されがちでした．近年，コンピュータの発達に伴って注目されるようになりました．RLSアルゴリズムは，最急降下法とSMIアルゴリズムの中間的なアルゴリズムで，最適な値にたどりつくまでの時間（収束時間）が短縮できるため，最も一般的に使われています．

● シストリック・アレイを用いる実装手法

スマート・アンテナのアルゴリズムは，回路実装に適さない逆行列や割り算などの演算が必要になります．また，高速処理を実現するには，演算回路を並列化した効率的な構造を考えなければなりません．このために古くから考案されたのがシストリック・アレイ（Systolic Array）構造です．まず，シストリック・アレイについて調べてみます．

1980年頃，H. T. KungとC. E. Leisersonは，VLSI実現向きのアーキテクチャが満たすべき条件として，

①少ない種類の単純な構造のセルを用いた回路構成
②単純で規則正しいデータと制御の流れ
③並列処理，パイプライン処理による高速計算
④入出力と処理の重畳

を挙げ，これらを満たすものとして，シストリック・アレイを提案しました．シストリック（Systolic）は，心臓収縮を意味する形容詞です．ちょうど血液が心臓の収縮に合わせて体内を循環するように，データがクロックに同期してプロセッサ・アレイ上を移動して処理を行います．上記の条件①および②は，回路の論理設計やレイアウト設計，アルゴリズム検証を容易にするために重要ですが，回路の拡張性の点からも重要です．

図8 シストリック・アレイの構造

大規模な回路を一つのチップ上に実現できないとき，同一のチップを必要個数だけ結合して一つの回路を構成できることが望ましい場合があります．その際，回路全体の入出力機構が，そのままチップ間の結合に使用できることが望ましいでしょう．

条件③および条件④は，高速計算を実現するために重要です．特に，処理に要する時間が比較的小さな場合は，回路からのデータの転送と処理を重畳し，入出力を含めた全体の処理時間を短くすることが重要になります．

シストリック・アレイは多数の同一あるいは少ない種類の簡単なプロセッシング・エレメント（PE）を規則正しく接続して構成されます．アレイの形状は，**図8**に示すように，1次元配列，2次元正方配列，2次元三角配列が基本的な構造となります．データはアレイへ逐次的に入出力され，アレイ上の多数のPEで並列に処理されます．これらの入出力および処理は，パイプライン化され，重畳して進められます．

シストリック・アレイおよびその上でのアルゴリズムであるシストリック・アルゴリズムは，以下の条件を満たします．

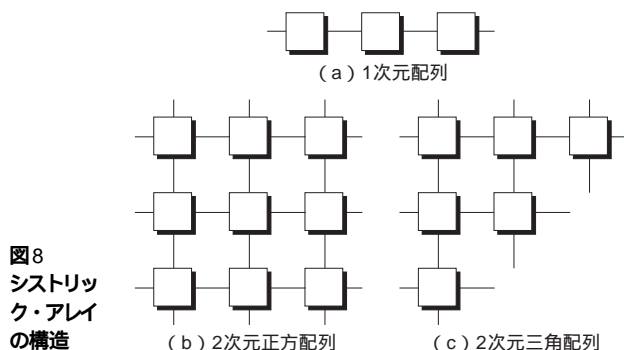
（1）均一構造

アレイは同一（あるいは少ない種類）の簡単なプロセシング・エレメント（PE）からなります．PEはセルあるいはシストリック・セルと呼ばれ，各セルは有限個のレジスタをもちます．四則演算などが1ステップ（単位時間）で実行されます．

（2）局所結合と局所通信

セルは隣接するセルとのみ直接接続されます（局所結合）．各セルが1ステップで直接通信できるのは，近傍のセルに限定し，これを局所通信と呼びます．アレイ上でのデータの送受信はすべて局所結合を通じて局所的に行われます（ほかに，すべてのセルに共通する配線として，電源とクロックがあるが，これらは局所的ではない）．

（3）逐次的入出力

初期状態において，どのセルもあらかじめデータをもっていません．処理されるデータはアレイへあらかじめ定められた一定の割合，例えば1ステップに1個あるいは2ステップに1個ずつ逐次入力されます．この割合を入力パイプライン・インターバルと呼びます．出力も同様に，逐次的に行わ

れ，出力の時間間隔を出力パイプライン・インターバルと呼びます．このような条件を満たすシストリック構造は，FPGAなどのハードウェア実装に適しているといえます．

3．FPGAを用いた回路設計

ここでは，FPGAを用いてMMSE法をシストリック構造で実装した例を紹介します．行列のQR分解（QR Decomposition）を用いた再帰的最小二乗法のRLSアルゴリズムは，高速な収束特性と数値的な安定性が知られており，今回はこのQRD-RLSをFPGAで実装することにしました．

図9にQRD-RLSシストリック・アレイに基づいた受信システムの概略を示します．RF（Radio Frequency）帯より到来してきた波は，アレイ・アンテナの各素子から受信されます．DBF（Digital Beam Forming）受信機にてIF（Intermediate Frequency）帯の信号へと変換され，A-D変換の後，信号はFPGAへと送られます．FPGAではDDC（Digital Down Conversion）でさらに複素ベースバンド信号に変換されます．QRD-RLSシストリック・アレイ・プロセッサによる重み係数の更新が行われ，干渉波や遅延波の影響が取り除かれたアレイ出力を得るという仕組みになっています[4]．FPGAは米国Altera社のStratix（EP1S40）を使用しました．

複素重み係数ベクトルは各アンテナからの受信信号に掛けられますが，FPGAのDSPブロックを用いると高速に動作する乗算器を実装できます．このような特徴を生かして，複素乗算器を実装しビーム形成処理にかかる配線とロジック回路を低減できます．

● QR分解型RLSシストリック・アレイ

MMSE法の制御アルゴリズムであるRLS法は，古くから，等化処理や適応フィルタなど，数多い無線応用に用いられています．これは，一般に次式のように優決定（Overdetermined；式が未知数より多い）の方程式の解として用いられます．アレイ・アンテナの場合，アレイの出力は，次のような関係式で表現されます（$m > N$）．

$$\begin{aligned} x_1(1)\omega_0 + x_2(1)\omega_1 + \cdots + x_N(1)\omega_{N-1} &= y(1) + e(1) \\ x_1(2)\omega_0 + x_2(2)\omega_1 + \cdots + x_N(2)\omega_{N-1} &= y(2) + e(2) \\ &\vdots \\ x_1(m)\omega_0 + x_2(m)\omega_1 + \cdots + x_N(m)\omega_{N-1} &= y(m) + e(m) \end{aligned} \quad \cdots(9)$$

ここで，mは時間インデックス，Nは素子数です．これはベクトル表記に書き直すと，

$$\boldsymbol{\omega}^{\mathrm{H}}\boldsymbol{x}(t) = y(t) + e(t) \quad \cdots\cdots(10)$$

のようになり，$e(t)$の平均二乗誤差を最小にする重みωを求める問題になります[1]．

図10にM＝4素子におけるQRD-RLSシストリック・アレイ・プロセッサを示します．丸いセルは境界セル，正方形のセルは内部セルに対応します（**図11**）．各行では，境界セルで回転角度が得られ，その角度に応じて内部セルで回転を行うことによってQR分解が行われます．最終的に各セルには，$\boldsymbol{R}(n)$と$\boldsymbol{u}(n)$の値が更新され保存されます．

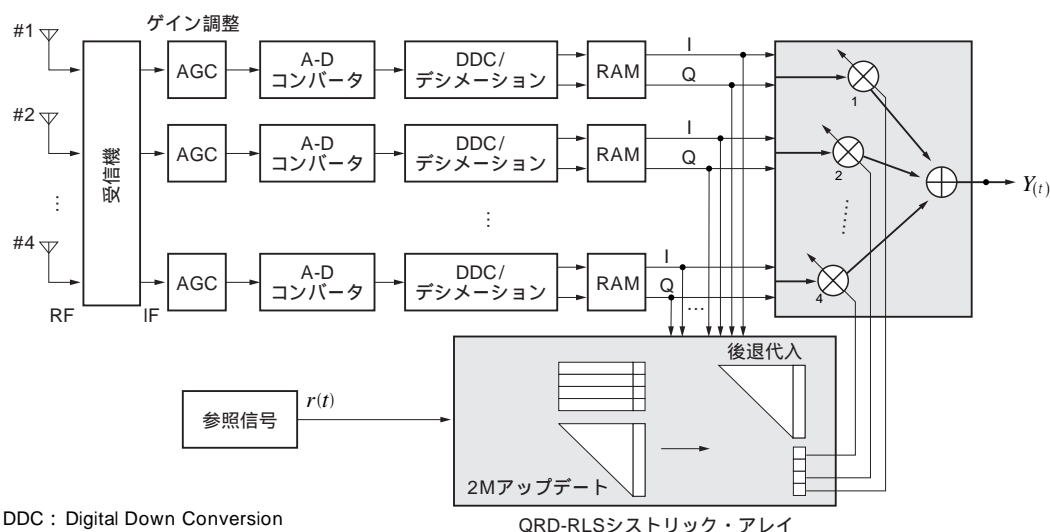



図9 QRD-RLSシストリック・アレイ（4素子アレイ・アンテナ）

図10 QR分解を用いるシストリック・アレイ・プロセッサ

図11　境界セルと内部セル

重み係数ベクトルωは次に示すような後退代入（Back Substitution）と呼ばれる方法によって求められます．QRD-RLSアルゴリズムの流れを**図12**に示します．

各セルは**図13**に示されるCPE（CORDIC Processor Element）によって処理が行われ，また，その処理は逐次形式のCORDICで構成されます．境界セルと内部セルは，CORDICにおけるベクトリング（Vectoring Mode）とローテーション・モード（Rotation Mode）にそれぞれ対応しています．

ベクトリング・モードでは，複素数x_{in}の虚数部がφ-CPEによって除去され実数となり，次にθ-CPE1によってx_{in}がゼロとなり，回転角度のφ，θがそれぞれ求められます．ローテーション・モードでは，これらの回転角度に応じて入力ベクトル$[\mathrm{Re}(x_{in}), \mathrm{Im}(x_{in})]^T$をφ-CPEによってφほど回転させて，次に$r$と$x$の実数部と虚数部をθ-CPE1とθ-CPE2でθほどそれぞれ回転させます

パイプライン化された計算ブロック（CPE）は，データを上から順に流し込むことにより効率的な計算が行われます．

図14は，4素子リニア・アレイ・アンテナにおいて，0°方向に所望波が，－40°方向から干渉波が到来する場合の

図12　QR分解シストリック・アレイにおける重み係数の計算

図13　CPE（CORDIC processor element）の構成（複素形式）

ビーム・パターンを表しています．所望波方向にメイン・ビームを向け干渉波方向にヌルを向けていることが確認できます．実装したRLSアルゴリズムが干渉波の影響を取り除くように動作していることが分かります．

リスト1に，2素子シストリック・アレイのトップ・レベルVHDL記述を示します．**図10**を参考に結線を確認してください．ベクトリング・ローテーション・モードのコンポーネント記述は省略しますが，pp.106-111のAppendixを参考にすれば，容易に実装できるでしょう．

参考・引用*文献

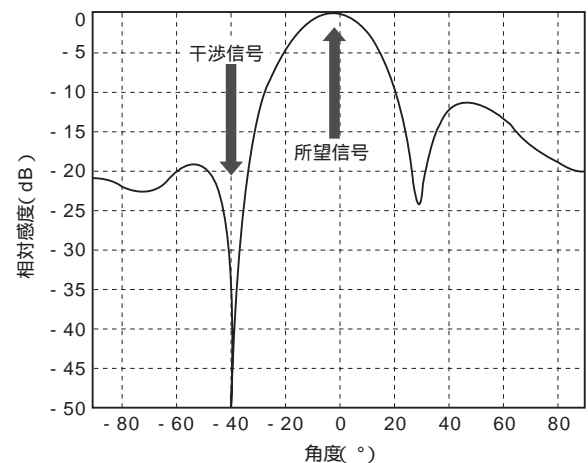
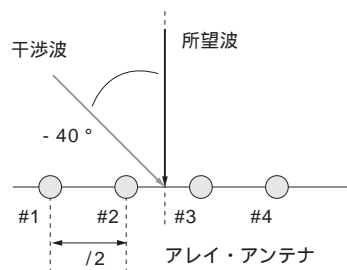
(1) Altera；White paper，“*Implementation of CORDIC-based QRD-RLS Algorithm on Altera Stratix FPGA with Embedded Nios Soft Processor Technology*”

(2) 菊間信良；アダプティブアンテナ技術，2003年，オーム社．

(3) 大鐘武雄，小川恭孝；アダプティブアレーと移動通信(1)－移動通信伝搬路への適用－，電子情報通信学会誌，Vol.81，No.12，pp.1254-1260，1998年12月．

リスト1　2素子シストリック・アレイのVHDL記述例

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.ALL;
USE IEEE.std_logic_UNSIGNED.all;

ENTITY systolic_2ele IS
  GENERIC (
    B            : Integer := 16
  );
  PORT (
    clk         : IN std_logic;
    reset       : IN std_logic;

    X11_Re, X11_Im   : IN std_logic_vector(B-1 downto 0);
    X12_Re, X12_Im   : IN std_logic_vector(B-1 downto 0);
    Y1_Re, Y1_Im     : IN std_logic_vector(B-1 downto 0);

    R11, R22 : OUT std_logic_vector(B-1 downto 0);

    R12_Re, R12_Im : OUT std_logic_vector(B-1 downto 0);

    U1_Re, U1_Im    : OUT std_logic_vector(B-1 downto 0);
    U2_Re, U2_Im    : OUT std_logic_vector(B-1 downto 0)
   );
END ENTITY systolic_2ele;

ARCHITECTURE rtl OF systolic_2ele IS

COMPONENT rotation_mode
  GENERIC (
    B            : Integer := 16
  );
  PORT (
    clk       : IN std_logic;
    reset     : IN std_logic;

    x_in      : IN std_logic_vector(B-1 downto 0);
    y_in      : IN std_logic_vector(B-1 downto 0);
    phi_in     : IN std_logic_vector(B downto 0);
    theta_in     : IN std_logic_vector(B downto 0);

    r_re      : OUT std_logic_vector(B-1 downto 0);
    r_im      : OUT std_logic_vector(B-1 downto 0);
    x_out     : OUT std_logic_vector(B-1 downto 0);
    y_out     : OUT std_logic_vector(B-1 downto 0);
    phi_out     : OUT std_logic_vector(B downto 0);
    theta_out     : OUT std_logic_vector(B downto 0)

  );
END COMPONENT;

COMPONENT vectoring_mode
  GENERIC (
    B            : Integer := 16
  );
  PORT (
    clk       : IN std_logic;
    reset     : IN std_logic;

    x_in      : IN std_logic_vector(B-1 downto 0);
    y_in      : IN std_logic_vector(B-1 downto 0);

    r_re      : OUT std_logic_vector(B-1 downto 0);
    phi_out    : OUT std_logic_vector(B downto 0);
    theta_out    : OUT std_logic_vector(B downto 0)

  );
END COMPONENT;

SIGNAL P12,Q12            : std_logic_vector(B downto 0);
SIGNAL P13,P23,Q13,Q23    : std_logic_vector(B downto 0);
SIGNAL P14,P24,Q14,Q24    : std_logic_vector(B downto 0);

SIGNAL X22_Re,X22_Im    : std_logic_vector(B-1 downto 0);
SIGNAL Y2_Re,Y2_Im,Y3_Re,Y3_Im          :
                              std_logic_vector(B-1 downto 0);

BEGIN

   BOUNDARY1 : vectoring_mode
      GENERIC MAP(B => B)
      PORT MAP (clk, reset, X11_Re, X11_Im, R11, P12,
                                                   Q12);

   INTERNAL1 : rotation_mode
      GENERIC MAP(B => B)
      PORT MAP (clk, reset, X12_Re, X12_Im, P12, Q12,
               R12_Re, R12_Im, X22_Re, X22_Im, P13, Q13);

   BOUNDARY2 : vectoring_mode
      GENERIC MAP(B => B)
      PORT MAP (clk, reset, X22_Re, X22_Im, R22, P23,
                                                   Q23);

   INTERNAL2 : rotation_mode
      GENERIC MAP(B => B)
      PORT MAP (clk, reset, Y1_Re, Y1_Im, P13, Q13,
                   U1_Re, U1_Im, Y2_Re, Y2_Im, P14, Q14);

   INTERNAL3 : rotation_mode
      GENERIC MAP(B => B)
      PORT MAP (clk, reset, Y2_Re, Y2_Im, P23, Q23,
                   U2_Re, U2_Im, Y3_Re, Y3_Im, P24, Q24);


END rtl;
```

図14
適応ビーム形成例（4素子アレイ・アンテナ）

(4) Yoshiaki Yokoyama, Minseok Kim and Hiroyuki Arai; "*Implementation of Systolic RLS adaptive array using FPGA and Its Performance Evaluation,*" 2006 IEEE Vehicular Technology Conference (VTC 2006 fall), Montreal, CA, Sept. 2006.

Minseok Kim
東京工業大学

第3章
Appendix

通信回路に役立つ CORDICアルゴリズム

Minseok Kim

CORDIC（Coordinate Rotation Digital Computer）アルゴリズムは，三角関数，四則演算，極座標系と直交座標系の変換などの計算のために，米国のJack E. Volder氏により1959年に考案されました[(1)].

CORDICアルゴリズムは，ビット・シフト（Bit Shift）と足し算（Add）のみで，ある回転角度に対するベクトル回転を反復的に行えます．回路実装が容易であることから，さまざまな信号処理回路に適用されています．例えば，ディジタル検波回路でのNCO（数値制御発信器）やDDS（Direct Digital Synthesizer），固有値/特異値分解，各種Lattice フィルタ，DFT（Discrete Fourier Transform）/DCT（Discrete Cosine Transform）など，いろいろな通信回路において非常に役に立つものです．ここではその理論を解説し，VHDLで回路を設計してみます．

1．CORDIC法の原理

図1に示すように，CORDICアルゴリズムは基本的にある座標$V=[x,\ y]$から$V'=[x',\ y']$にある角度φで回転する動作を反復します．これを式で表現すると，回転の公式を使い次式のように書けます．

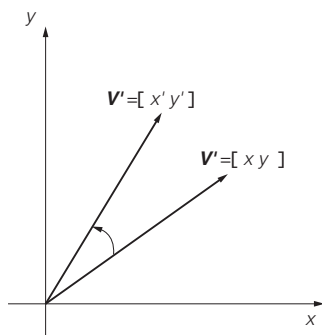



図1
角度φに対するベクトル回転

$$V'=\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}\cos\phi & -\sin\phi\\ \sin\phi & \cos\phi\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix}=\begin{bmatrix}x\cdot\cos\phi-y\cdot\sin\phi\\ x\cdot\sin\phi+y\cdot\cos\phi\end{bmatrix} \quad \cdots\cdots(1)$$

これをもっと整理したのが次式になります．

$$\begin{bmatrix}x'\\y'\end{bmatrix}=\cos\phi\cdot\begin{bmatrix}x-y\cdot\tan\phi\\ y+x\cdot\tan\phi\end{bmatrix} \quad \cdots\cdots(2)$$

この式でcos φはtan φで変形できます．CORDICの考え方は，このtan φを離散化することに着目します．要するに，回転角度φを離散化した値の反復回転を行い，その回転方向による符号付き和で表現されます．tan φは以下のように離散化されます．

$$\tan\phi\equiv 2^{-i} \qquad \phi=\tan^{-1}2^{-i} \quad \cdots\cdots(3)$$

これは，最初のベクトルから45°回転，14°回転，7°回転，…のように（大まかな回転から段々細かく）回転方向を制御しながらφに近づいていきます．bビット・ワード長をもつ入力ベクトル$V=[x,\ y]$を，角度φで回転する場合，$b+1$回の反復が必要になり，固定小数点演算の場合，ワード長でその計算精度が決まります．

cos φをφ＝$\tan^{-1}(2^{-k})$の関係を用いると以下のようになり，

$$\cos\phi=\frac{1}{\sqrt{1+\tan^2\phi}}=\frac{1}{\sqrt{1+2^{-2k}}} \quad \cdots\cdots(4)$$

反復式に書き直すと次式のようになります．

$$\begin{bmatrix}x_{i+1}\\y_{i+1}\end{bmatrix}=K_i\cdot\begin{bmatrix}x_i-y_i\cdot d_i\cdot 2^{-i}\\ y_i-x_i\cdot d_i\cdot 2^{-i}\end{bmatrix} \quad \cdots\cdots(5)$$

ここで，$d=\pm1$（回転方向を示す）とK_iは回転により生じる利得（Processing Gain）であり，次のようになります．

$$K_i = \prod_{k=0}^{i} \frac{1}{\sqrt{1+2^{-2k}}} \quad \cdots\cdots(6)$$

これは，kにより決定される固定の値なので，ROM（Read Only Memory）などにあらかじめ計算しておくことができます．また，それぞれの回転においてゲイン調整をすることではなく，最終的なゲイン$|K|$で調整することで計算量を低減できます．

図2に反復回転によるベクトル回転の様子を示します．

d_iで表される回転方向は，角度アキュムレータ（Accumulator）による第三の差分関係式から定義されます．

$$z_{i+1} = z_i - d_i \cdot \tan^{-1}(2^{-i}) \quad \cdots\cdots(7)$$

ここで，角度の値はLUT（Look-up Table）を用いて実現されます．

では，CORDICアルゴリズムにおける二つの基本モードについて調べてみましょう．まず，Volder氏により名づけられたローテーション・モード（Rotation Mode）とベクトリング（Vectoring Mode）です．ローテーション・モードでは，角度アキュムレータを回転角度φとして初期化します．各回転ステップにおいて，角度アキュムレータを小さくするように次のステップでの回転方向を決めます．このような動作は次式のように表現されます．

$$\begin{cases} x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} = z_i - d_i \cdot \tan^{-1}\left(2^{-i}\right) \end{cases} \quad \cdots\cdots(8)$$

（$z_i < 0$のとき$d_i = -1$，それ以外のとき$d_i = +1$）

このようなステップをn回繰り返すことで，以下のような角度φのベクトル回転が得られます．

$$\begin{aligned} x_n &= A_n(x_0 \cos z_0 - y_0 \sin z_0) \\ y_n &= A_n(y_0 \cos z_0 + x_0 \sin z_0) \\ z_n &= 0 \\ A_n &= \prod_{i=0}^{n} \sqrt{1+2^{-2i}} \end{aligned} \quad \cdots\cdots(9)$$

次に，ベクトリング・モードの場合，入力ベクトルをx軸上（$y = 0$）になるまで回転します．この結果，xの値はベクトルの大きさになり，zの値は入力ベクトルの位相角になります．ベクトリング・モードは，各ステップにおいてyの値を小さくするように次のステップでの回転方向を決めます．ローテーション・モードと同様にこの動作は次式で表現されます．

$$\begin{cases} x_{i+1} = x_i - y_i \cdot d_i \cdot 2^{-i} \\ y_{i+1} = y_i + x_i \cdot d_i \cdot 2^{-i} \\ z_{i+1} = z_i - d_i \cdot \tan^{-1}\left(2^{-i}\right) \end{cases} \quad \cdots\cdots(10)$$

（$y_i < 0$ のとき $d_i = +1$，それ以外のとき$d_i = -1$）

また，計算ステップをn回繰り返すことで，以下のような結果が得られます．

$$\begin{aligned} x_n &= A_n \sqrt{{x_0}^2 + {y_0}^2} \\ y_n &= 0 \\ z_n &= z_0 + \tan^{-1}\left(y_0 / x_0\right) \\ A_n &= \prod_{i=0}^{n} \sqrt{1+2^{-2i}} \end{aligned} \quad \cdots\cdots(11)$$

ローテーション・モードやベクトリング・モードは，最初のtanの値が2^0（45°）になることから，$-\pi/2 < \phi < \pi/2$の範囲で制限されます．

2．回路設計

CORDICアルゴリズムはビット・シフトと足し算のみで実現でき，回路化が容易であると知られています．CORDICアルゴリズムを実装する方法は，要求性能や回路リソース別にいくつか上げられ，シリアル加算器を使うかどうかによりビット・シリアル/ビット・パラレル（Bit-Serial/Bit-Parallel），また，計算ステップを回路で反復的に再利用するかどうかにより展開/反復（Unrolled/Iterative）で分類できます．

図3に，性能がもっとも優れたBit-Parallel Unrolled構

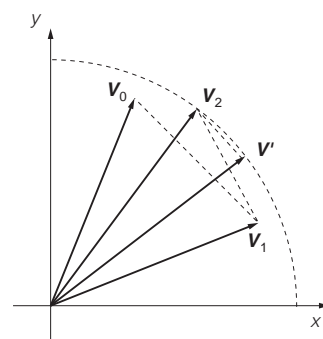



図2　ベクトル回転を繰り返す

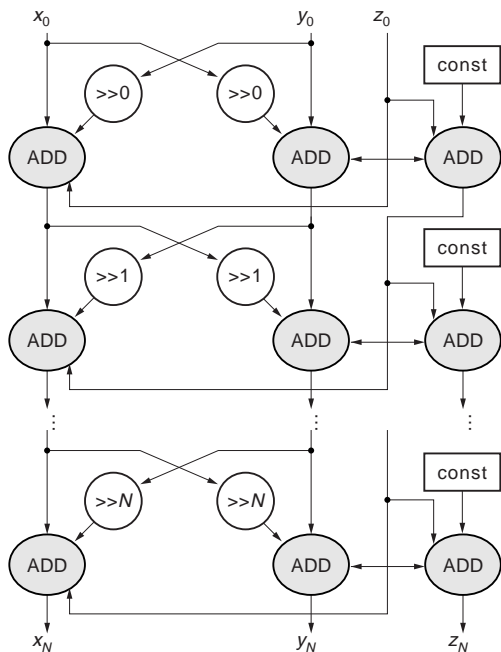



図3　Unrolled CORDIC構造

造を示します．

CORDICアルゴリズムでは，利得の調整（スケーリング）は必ず行わなければなりません．前述のようにROMなどを用いてあらかじめ計算しておけるため，計算負荷を低減できます．しかし，その値を用いた正規化には割り算や平方根演算を必要とするため，回路化が困難となります．この問題は，ダブル回転法により解決できます．

ダブル回転法は，角度φで回転するのではなく，次式のように角度φ/2を2回回転することを意味します．

$$\begin{cases} x_{k+1} = \left(1-2^{-2k}\right)x_k - y_k \cdot d_k \cdot 2^{-k+1} \\ y_{k+1} = \left(1-2^{-2k}\right)y_k - x_k \cdot d_k \cdot 2^{-k+1} \\ z_{k+1} = z_k - d_k \cdot \tan^{-1}\left(2^{-i}\right) \end{cases} \quad \cdots\cdots\cdots\cdots\cdots (12)$$

（$z_k < 0$のとき$d_k = -1$，それ以外のとき$d_k = +1$）

これは，各ステップで四つのビット・シフトと六つの足し算で実現されます．しかし，利得は平方根なしの状態に

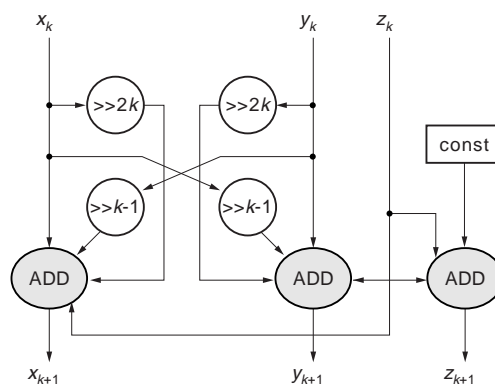



図4　k番目のダブル回転ステップ

なり，さらに固定小数点における与えられた精度で次のように近似することで，割り算の実装を避けられます[(2)]．

$$K_k = \prod_{i=0}^{k} \frac{1}{1+2^{-2i}} \approx \frac{1}{2}\prod_{i=0}^{k/4}\left(1-2^{-(4i-2)}\right) \quad \cdots\cdots\cdots\cdots\cdots (13)$$

このことから，利得のスケーリングをビット・シフトと足し算で行えます．**図4**にk番目のダブル回転ステップのブロック図を示します．

リスト1と**リスト2**では，回路の効率化を図るダブル回転CORDICアルゴリズムを採用したVHDL回路設計例を示します．入出力は16ビットとして設計しましたが，利得を考慮して18ビットにしてあります．ローテーション・モードとベクトリング・モードともに，最後の部分でスケーリングを行っています．

参考・引用*文献

(1) Volder J. E. ; "*The CORDIC trigonometric computing technique*," IRE Trans action Electronic Computing, Vol. EC-8, 1959.
(2) Jurgen Gotze, Steffen Paul and Matthias Sauer ; "*An Efficient Jacobi-like Algorithm for Parallel Eigenvalue Computation*", IEEE Trans action on Computer, Vol. 42, No. 9, Sep. 1993

Minseok Kim
東京工業大学

リスト1　CORDICローテーション・モードのVHDL記述

```
--*************************************--
--     CORDIC BIT-PARALLEL ROTATION
--=====================================--
--  FEATURES
-- 12bits Accuracy
-- 12bits I/O
-- 12bits atan LUT
-- twice rotation of theta/2 for scaling without sqrt

-- theta is 16bits accuracy %
--  pi   -->
--  pi/2 --> 0111 1111 1111 1111 ( 32767)
--  0    --> 0000 0000 0000 0000 (     0)
-- -pi/2 --> 1000 0000 0000 0000 (-32768)

-- d(rad)--> if d>0, fix(d/(pi/2)*( 2^(15-1)-1))(16bit)
--           if d<0, fix(d/(pi/2)*(-2^(15-1)  ))(16bit)

-- x, y are 16-bit accuracy but 18-bit word adding 2bits %
--  1 --> 00 0111 1111 1111 1111 ( 32767)
-- -1 --> 11 1000 0000 0000 0000 (-32768)

--    x(k+1) = (1-2^(-2k))*x(k) - d*2^(-k+1)*y(k)
--    y(k+1) = (1-2^(-2k))*y(k) + d*2^(-k+1)*x(k)
--    z(k+1) = theta - d*arctan(2^-k)

-- Input z : z/2
--*************************************--
LIBRARY ieee;
USE  ieee.std_logic_1164.all;

PACKAGE cordic_package IS
  FUNCTION bitlshift(x:std_logic_vector; n:natural)
                              RETURN std_logic_vector;
  FUNCTION bitrshift(x:std_logic_vector; n:natural)
                              RETURN std_logic_vector;
  FUNCTION decisign(x:std_logic_vector; dsign:std_logic)
                              RETURN std_logic_vector;
END cordic_package;

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_ARITH.all;
USE IEEE.std_logic_SIGNED.all;

PACKAGE BODY cordic_package IS
  FUNCTION bitlshift(x:std_logic_vector; n:natural)
    RETURN std_logic_vector IS
    VARIABLE z : std_logic_vector(x'range);
    BEGIN
      z(x'high) := x(x'high);
      FOR i IN x'high - 1 DOWNTO n LOOP
        z(i) := x(i-n);
      END LOOP;
      FOR i IN n-1 DOWNTO 0 LOOP
        z(i) := '0';
      END LOOP;
      RETURN z;
    END;

  FUNCTION bitrshift(x:std_logic_vector; n:natural)
    RETURN std_logic_vector IS
    VARIABLE z : std_logic_vector(x'range);
    CONSTANT lo: integer := x'high - n + 1;
    BEGIN
      IF lo > 0 THEN
        FOR i IN x'high DOWNTO lo LOOP
          z(i) := x(x'high);
        END LOOP;
        FOR i IN lo - 1 DOWNTO 0 LOOP
          z(i) := x(i+n);
        END LOOP;
      ELSE
        FOR i IN x'high DOWNTO 0 LOOP
          z(i) := x(x'high);
        END LOOP;
      END IF;
      RETURN z;
    END;

  FUNCTION decisign(x:std_logic_vector; dsign:std_logic)
    RETURN std_logic_vector IS
    VARIABLE z:std_logic_vector(x'range);
    BEGIN
      IF dsign='1' THEN -- 1 is negative
        z := -x;
      ELSE              -- 0 is positive
        z := x;
      END IF;
      RETURN(z);
    END;
END cordic_package;

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_ARITH.all;
USE IEEE.std_logic_SIGNED.all;
USE work.cordic_package.ALL;

ENTITY cordic_bp_rot IS
  GENERIC (
    B         : Integer := 16;
    PIPENO    : natural := 1);
  PORT (
    clk       : IN std_logic;
    ena       : IN std_logic;

    x_in      : IN std_logic_vector(B+1 downto 0);
    y_in      : IN std_logic_vector(B+1 downto 0);
    z_in      : IN std_logic_vector(B-1 downto 0);

    x_out     : OUT std_logic_vector(B+1 downto 0);
    y_out     : OUT std_logic_vector(B+1 downto 0);
    z_out     : OUT std_logic_vector(B-1 downto 0)
  );
END ENTITY cordic_bp_rot;


ARCHITECTURE rtl OF cordic_bp_rot IS
-- 14bits
  TYPE LUTARCTANTAB IS ARRAY(0 to B-1) OF
                              std_logic_vector(B-1 downto 0);
  CONSTANT ARCTANTAB:LUTARCTANTAB :=
  ( "01000000000000", "00100101110010", "00010011111101",
    "00001010001000", "00000101000101", "00000010100011",
    "00000001010001", "00000000101001", "00000000010100",
    "00000000001010", "00000000000101", "00000000000011",
    "00000000000001", "00000000000001");

  -- (1024, 604, 319, 162, 81, 41, 20, 10, 5, 3, 1, 1);
  -- (16384, 9672, 5110, 2594, 1302, 652, 326, 163, 81,
                                41, 20, 10, 5, 3, 1, 1)

  SIGNAL xreg, yreg      : std_logic_vector(B+1 downto 0);
  SIGNAL zreg            : std_logic_vector(B-1 downto 0);
  SIGNAL sx, sy          : std_logic_vector(B+1 downto 0);
  SIGNAL sz              : std_logic_vector(B-1 downto 0);

  BEGIN

    PROCESS (clk,ena) IS
      BEGIN
        IF clk'event AND clk = '1' THEN
          IF ena = '1' THEN
            x_out <= xreg;
            y_out <= yreg;
            z_out <= zreg;
          END IF;
        END IF;
      END PROCESS;

--    x(k+1) = (1-2^(-2k))*x(k) - d*2*2^(-k)*y(k)
--    y(k+1) = (1-2^(-2k))*y(k) + d*2*2^(-k)*x(k)
--    z(k+1) = theta - d*arctan(2^-k)

      sx <= decisign(bitrshift(x_in, PIPENO), z_in(B-1));
      sy <= decisign(bitrshift(y_in, PIPENO), z_in(B-1));
      sz <= decisign(ARCTANTAB(PIPENO), z_in(B-1));

      xreg <= (( x_in - bitrshift(x_in, 2*PIPENO)) -
                                        bitlshift(sy, 1));
      yreg <= (( y_in - bitrshift(y_in, 2*PIPENO)) +
```

リスト1　CORDICローテーション・モードのVHDL記述（つづき）

```
                                    bitlshift(sx, 1));
      zreg <= z_in - sz;

  END ARCHITECTURE rtl;


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_ARITH.all;
USE IEEE.std_logic_SIGNED.all;
USE work.cordic_package.ALL;

ENTITY cordic_rotation IS
  GENERIC (
    B           : Integer := 16;
    NOofPIPES   : natural := 16
  );
  PORT (
    clk      : IN std_logic;
    reset    : IN std_logic;
    ena      : IN std_logic;

    x_in     : IN std_logic_vector(B-1 downto 0);
    y_in     : IN std_logic_vector(B-1 downto 0);
    z_in     : IN std_logic_vector(B-1 downto 0);

    x_out    : OUT std_logic_vector(B-1 downto 0);
    y_out    : OUT std_logic_vector(B-1 downto 0)
  );
END ENTITY cordic_rotation;

ARCHITECTURE rtl OF cordic_rotation IS

  COMPONENT cordic_bp_rot IS
    GENERIC (
      B         : Integer := 16;
      PIPENO    : natural := 1);
    PORT (
      clk      : IN std_logic;
      ena      : IN std_logic;

      x_in     : IN std_logic_vector(B+1 downto 0);
      y_in     : IN std_logic_vector(B+1 downto 0);
      z_in     : IN std_logic_vector(B-1 downto 0);

      x_out    : OUT std_logic_vector(B+1 downto 0);
      y_out    : OUT std_logic_vector(B+1 downto 0);
      z_out    : OUT std_logic_vector(B-1 downto 0)
    );
  END COMPONENT;

  TYPE TARRAY IS ARRAY(0 to B) OF std_logic_vector(B-1
                                              downto 0);
  SIGNAL zz                                 : TARRAY;

  TYPE OUTARRAY IS ARRAY(0 to B) OF std_logic_vector(B+1
                                              downto 0);
  SIGNAL xx                                 : OUTARRAY;
  SIGNAL yy                                 : OUTARRAY;

  SIGNAL   x_scale_1_3, x_scale_7_9, x_scale_11_13    :
                         std_logic_vector(B+1 downto 0);
  SIGNAL   y_scale_1_3, y_scale_7_9, y_scale_11_13    :
                         std_logic_vector(B+1 downto 0);

  SIGNAL   x_scale    : std_logic_vector(B+1 downto 0);
  SIGNAL   y_scale    : std_logic_vector(B+1 downto 0);

  BEGIN

    CORLOOP :
      FOR k in 1 TO B GENERATE
        cor : cordic_bp_rot
          GENERIC MAP(B => B, PIPENO => k-1)
          PORT MAP (clk, ena, xx(k-1), yy(k-1), zz(k-1),
                              xx(k), yy(k), zz(k));
      END GENERATE CORLOOP;

      xx(0) <= x_in(B-1) & x_in(B-1)& x_in(B-1) &
                                  x_in(B-2 downto 0);
      yy(0) <= y_in(B-1) & y_in(B-1)& y_in(B-1) &
                                  y_in(B-2 downto 0);
      zz(0) <= bitrshift(z_in, 1);

-- scaling

    x_scale_1_3   <= bitrshift(xx(B), 1) -
                                 bitrshift(xx(B), 3);
    x_scale_7_9   <= bitrshift(xx(B), 7) -
                                 bitrshift(xx(B), 9);
    x_scale_11_13 <= bitrshift(xx(B),11) -
                                 bitrshift(xx(B),13);

    y_scale_1_3   <= bitrshift(yy(B), 1) -
                                 bitrshift(yy(B), 3);
    y_scale_7_9   <= bitrshift(yy(B), 7) -
                                 bitrshift(yy(B), 9);
    y_scale_11_13 <= bitrshift(yy(B),11) -
                                 bitrshift(yy(B),13);

    x_scale <= (x_scale_1_3 - x_scale_7_9) -
               (x_scale_11_13 + bitrshift(xx(B),15)) ;
    y_scale <= (y_scale_1_3 - y_scale_7_9) -
               (y_scale_11_13 + bitrshift(yy(B),15)) ;

    x_out <= (x_scale(B+1) & x_scale(B-2 downto 0));
    y_out <= (y_scale(B+1) & y_scale(B-2 downto 0));

  END ARCHITECTURE rtl;
```

リスト2 CORDICベクトリング・モードのVHDL記述

```
--**************************************--
--      CORDIC BIT-PARALLEL VECTORING
--======================================--

LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_ARITH.all;
USE IEEE.std_logic_SIGNED.all;
USE work.cordic_package.ALL;

ENTITY cordic_bp_vec IS
  GENERIC (
    B         : Integer := 16;
    PIPENO    : natural := 1);
  PORT (
    clk       : IN std_logic;
    ena       : IN std_logic;

    x_in      : IN std_logic_vector(B+1 downto 0);
    y_in      : IN std_logic_vector(B+1 downto 0);
    z_in      : IN std_logic_vector(B-1 downto 0);

    x_out     : OUT std_logic_vector(B+1 downto 0);
    y_out     : OUT std_logic_vector(B+1 downto 0);
    z_out     : OUT std_logic_vector(B-1 downto 0)
  );
END ENTITY cordic_bp_vec;


ARCHITECTURE rtl OF cordic_bp_vec IS

-- 14bits
  TYPE LUTARCTANTAB IS ARRAY(0 to B-1) OF
                         std_logic_vector(B-1 downto 0);
  CONSTANT ARCTANTAB:LUTARCTANTAB :=
  ( "01000000000000", "00100101110010", "00010011111101",
    "00001010001000", "00000101000101", "00000010100011",
    "00000001010001", "00000000101001", "00000000010100",
    "00000000001010", "00000000000101", "00000000000011",
    "00000000000001", "00000000000001");

  -- (1024, 604, 319, 162, 81, 41, 20, 10, 5, 3, 1, 1);
  -- (16384, 9672, 5110, 2594, 1302, 652, 326, 163, 81,
                              41, 20, 10, 5, 3, 1, 1)

  SIGNAL xreg, yreg                        :
                         std_logic_vector(B+1 downto 0);
  SIGNAL zreg                              :
                         std_logic_vector(B-1 downto 0);
  SIGNAL sx, sy                            :
                         std_logic_vector(B+1 downto 0);
  SIGNAL sz                                :
                         std_logic_vector(B-1 downto 0);

  BEGIN

    PROCESS (clk,ena) IS
      BEGIN
        IF clk'event AND clk = '1' THEN
          IF ena = '1' THEN
            x_out <= xreg;
            y_out <= yreg;
            z_out <= zreg;
          END IF;
        END IF;
      END PROCESS;

--    x(k+1) = (1-2^(-2k))*x(k) - d*2*2^(-k)*y(k)
--    y(k+1) = (1-2^(-2k))*y(k) + d*2*2^(-k)*x(k)
--    z(k+1) = theta - d*arctan(2^-k)

      sx <= decisign(bitrshift(x_in, PIPENO), NOT
                                          y_in(B-1));
      sy <= decisign(bitrshift(y_in, PIPENO), NOT
                                          y_in(B-1));
      sz <= decisign(ARCTANTAB(PIPENO), NOT y_in(B-1));

      xreg <= (( x_in - bitrshift(x_in, 2*PIPENO)) -
                                 bitlshift(sy, 1));
      yreg <= (( y_in - bitrshift(y_in, 2*PIPENO)) +
                                 bitlshift(sx, 1));
      zreg <= Z_In - sz;

  END ARCHITECTURE rtl;


LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_ARITH.all;
USE IEEE.std_logic_SIGNED.all;
USE work.cordic_package.ALL;

ENTITY cordic_arctan IS
  GENERIC (
    B            : Integer := 16;
    NOofPIPES    : natural := 16
  );
  PORT (
    clk       : IN std_logic;
    reset     : IN std_logic;
    ena       : IN std_logic;

    x_in      : IN std_logic_vector(B-1 downto 0);
    y_in      : IN std_logic_vector(B-1 downto 0);

    z_out     : OUT std_logic_vector(B-1 downto 0)
  );
END ENTITY cordic_arctan;

ARCHITECTURE rtl OF cordic_arctan IS

  COMPONENT cordic_bp_vec IS
    GENERIC (
      B         : Integer := 16;
      PIPENO    : natural := 1);
    PORT (
      clk       : IN std_logic;
      ena       : IN std_logic;

      x_in      : IN std_logic_vector(B+1 downto 0);
      y_in      : IN std_logic_vector(B+1 downto 0);
      z_in      : IN std_logic_vector(B-1 downto 0);

      x_out     : OUT std_logic_vector(B+1 downto 0);
      y_out     : OUT std_logic_vector(B+1 downto 0);
      z_out     : OUT std_logic_vector(B-1 downto 0)
    );
  END COMPONENT;

  TYPE TARRAY IS ARRAY(0 to B) OF std_logic_vector(B-1
                                           downto 0);
  SIGNAL zz                                 : TARRAY;

  TYPE OUTARRAY IS ARRAY(0 to B) OF std_logic_vector(B+1
                                           downto 0);
  SIGNAL xx                                 : OUTARRAY;
  SIGNAL yy                                 : OUTARRAY;

  BEGIN

    CORLOOP :
      FOR k in 1 TO B GENERATE
        cor : cordic_bp_vec
          GENERIC MAP(B => B, PIPENO => k-1)
          PORT MAP (clk, ena, xx(k-1), yy(k-1), zz(k-1),
                                   xx(k), yy(k), zz(k));
      END GENERATE CORLOOP;

      xx(0) <= x_in(B-1) & x_in(B-1)& x_in(B-1) &
                                     x_in(B-2 downto 0);
      yy(0) <= y_in(B-1) & y_in(B-1)& y_in(B-1) &
                                     y_in(B-2 downto 0);
      zz(0) <= (OTHERS=>'0');

      z_out <= zz(B); --bitlshift(zz(B), 1);

  END ARCHITECTURE rtl;
```